TM 841143

841143 ØØ1N

NAVAL UNDERWATER SYSTEMS CENTER
NEW LONDON LABORATORY
NEW LONDON, CONNECTICUT 06320

Technical Memorandum

FAST MOVING AVERAGE RECURSIVE LEAST MEAN SQUARE FIT

Date: 30 September 1984

Prepared by: Laurence C. Ng
Lawrence C. Ng

[illegible]
Paul R. Lambert
Passive Sonar Division
Submarine Sonar Department

Report Documentation Page			Form Approved OMB No. 0704-0188

1. REPORT DATE **30 SEP 1984**	2. REPORT TYPE **Technical Memo**	3. DATES COVERED **30-09-1984 to 30-09-1984**

4. TITLE AND SUBTITLE **Fast Moving Average Recursive Least Mean Square Fit**	5a. CONTRACT NUMBER
	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S) **Lawrence Ng; Paul Lambert**	5d. PROJECT NUMBER **A75210**
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Underwater Systems Center,New London,CT,06320**	8. PERFORMING ORGANIZATION REPORT NUMBER **TM 841143**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) **Naval Material Command**	10. SPONSOR/MONITOR'S ACRONYM(S) **MAT-058**
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
NUWC2015

14. ABSTRACT
A new approach is developed to reduce the computational complexity of a moving average Least Mean Square Fit (LMSF) procedure. For a long data window, a traditional batch approach would result in a large number of multiplication and add operations {i.e., an order N, where N is the window length). This memorandum shows that the moving average batch LMSF procedure could be made equivalent to a recursive process with identical filter memory length but at an order of reduction in computation load. The increase in speed due to reduced computation could make the moving average LMSF procedure competitive for many real-time processing applications.

15. SUBJECT TERMS
LMSF; Least Mean Square Fit

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**41**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

cy001

ABSTRACT

A new approach is developed to reduce the computational complexity of a moving average Least Mean Square Fit (LMSF) procedure. For a long data window, a traditional batch approach would result in a large number of multiplication and add operations (i.e., an order N, where N is the window length). This memorandum shows that the moving average batch LMSF procedure could be made equivalent to a recursive process with identical filter memory length but at an order of reduction in computation load. The increase in speed due to reduced computation could make the moving average LMSF procedure competitive for many real-time processing applications.

ADMINISTRATIVE INFORMATION

This memorandum was prepared under NUSC Project No. A75210, "Integrated Time Delay Estimation and Tracking", Principal Investigator, Dr. Lawrence C. Ng, (Code 3213); Sponsoring Activity, Naval Material Command (MAT-05B); Program Manager, Capt. Z. L. Newcomb.

The authors of this memorandum are located at the New London Laboratory, Naval Underwater Systems Center, New London, Connecticut 06320.

ACKNOWLEDGEMENTS

Technical review and comments made by Mr. R. LaTourette (Code 3213) on the numerical aspect of the recursive LMSF are highly appreciated.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	ii
1. INTRODUCTION	1
1.1 Problem Statement	3
1.2 Technical Objectives	7
1.3 Report Organization	7
2. MOVING AVERAGE RECURSIVE LINEAR LEAST SQUARE	7
2.1 Problem Formulation	7
2.2 Result Derivation	8
2.3 Result Analysis	11
3. MOVING AVERAGE RECURSIVE GENERAL LEAST SQUARE	11
3.1 Problem Formulation	11
3.2 Result Derivation	12
3.3 Result Analysis	15
4. NUMERICAL EVALUATION	16
4.1 Computational Method	16
4.2 Numerical Simulation	18
4.3 Speed Comparisons	23
4.4 Discussion of Results	28
5. SUMMARY AND CONCLUSIONS	29
REFERENCES	R-1
APPENDIX A LISTINGS OF COMPUTER PROGRAM	A-1

LIST OF FIGURES

Figure Page

1. Moving Average Least Mean Square Fit 2

2a. Recursive Moving Average First Order LMSF (Window Length = 50) 19
2b. Recursive Moving Average First Order LMSF Absolute Error . . 19

3a. Recursive Moving Average Second Order LMSF (Window Length = 50) 20
3b. Recursive Moving Average Second Order LMSF Absolute Error . . 20

4a. Recursive Moving Average Third Order LMSF (Window Length = 50) 21
4b. Recursive Moving Average Third Order LMSF Absolute Error . . 21

5a. Recursive Moving Average Fourth Order LMSF (Window Length = 50) 22
5b. Recursive Moving Average Fourth Order LMSF Absolute Error . . 22

6. CPU Time Versus Window Length Between Batch and Recursive First Order LMSF 24

7. CPU Time Versus Window Length Between Batch and Recursive Second Order LMSF 24

8. CPU Time Versus Window Length Between Batch and Recursive Third Order LMSF 25

9. CPU Time Versus Window Length Between Batch and Recursive Fourth Order LMSF 25

10. Recursive Over Batch Speed Ratio Versus Window Length (First Order LMSF) 26

11. Recursive Over Batch Speed Ratio Versus Window Length (Second Order LMSF) 26

12. Recursive Over Batch Speed Ratio Versus Window Length (Third Order LMSF) 27

13. Recursive Over Batch Speed Ratio Versus Window Length (Fourth Order LMSF) 27

1. INTRODUCTION

In processing a long stream of digital data, a polynomial moving average Least Mean Square Fit (LMSF) is sometimes used to provide smoothing or filtering of the noisy component imbedded in the data [1]. The LMSF has a finite duration memory which is determined by the length of the moving window. Figure 1 illustrates the successive polynomial LMSF obtained from an N-point window which slides over the input data sequence. These operations can be further explained as follows.

Let $z(i)$ for $i = 1, 2, \ldots$ be the measurement (data) sequence, and $y_n(k) = z(n - N + k)$ for $k = 1, 2, \ldots, N$ be an N-point sequence whose first data point is $z(n - N + 1)$ and the last data point is $z(n)$. Note that we assume $y_n(k) = 0$ for all $k < 1$. Let $\hat{y}_n(k)$ be the LMSF estimate based on the sequence $(y_n(k);\ k = 1, 2, \ldots, N)$. Then for a given k, the sequence $(\hat{y}_1(k), \hat{y}_2(k), \hat{y}_3(k), \ldots)$, obtained from a repeated LMSF operations, yields a smooth estimate of $y_n(k)$ for $n = 1, 2, 3, \ldots$. Note that for $1 \leq k < N$, LMSF performs a smoothing operation, and for $k = N$ the LMSF performs a filtering function if the measurement $z(n)$ is made available at the current time [2]. Also note that the LMSF estimates $\hat{y}_n(k)$ and $\hat{y}_{n+1}(k)$ contain the N-1 point overlapped data sequence $(z(n - N + 2), z(n - N + 3), \ldots, z(n))$. Simply stated, it is the recognition of the fact that moving average LMSF computations contain the overlapped sequence that ultimately allows the significant saving in computation. The reduction in computation is achieved via a successful development of a recursion relation between $\hat{y}_n(k)$ and $\hat{y}_{n+1}(k)$.

There are several advantages in using a moving average polynomial LMSF [3]. Foremost of these is that LMSF provides a stable operation since the LMSF polynomial coefficients are obtained from a bank of Finite Impulse Response (FIR) filters (shown later) operating on the data sequence. Secondly, the finite memory window assures us that bad data points which lie outside the window will have no effect on the resulting LMSF estimates.

On the other hand, there are also a number of disadvantages in the moving average LMSF application. First is that the sizable amount of memory storage

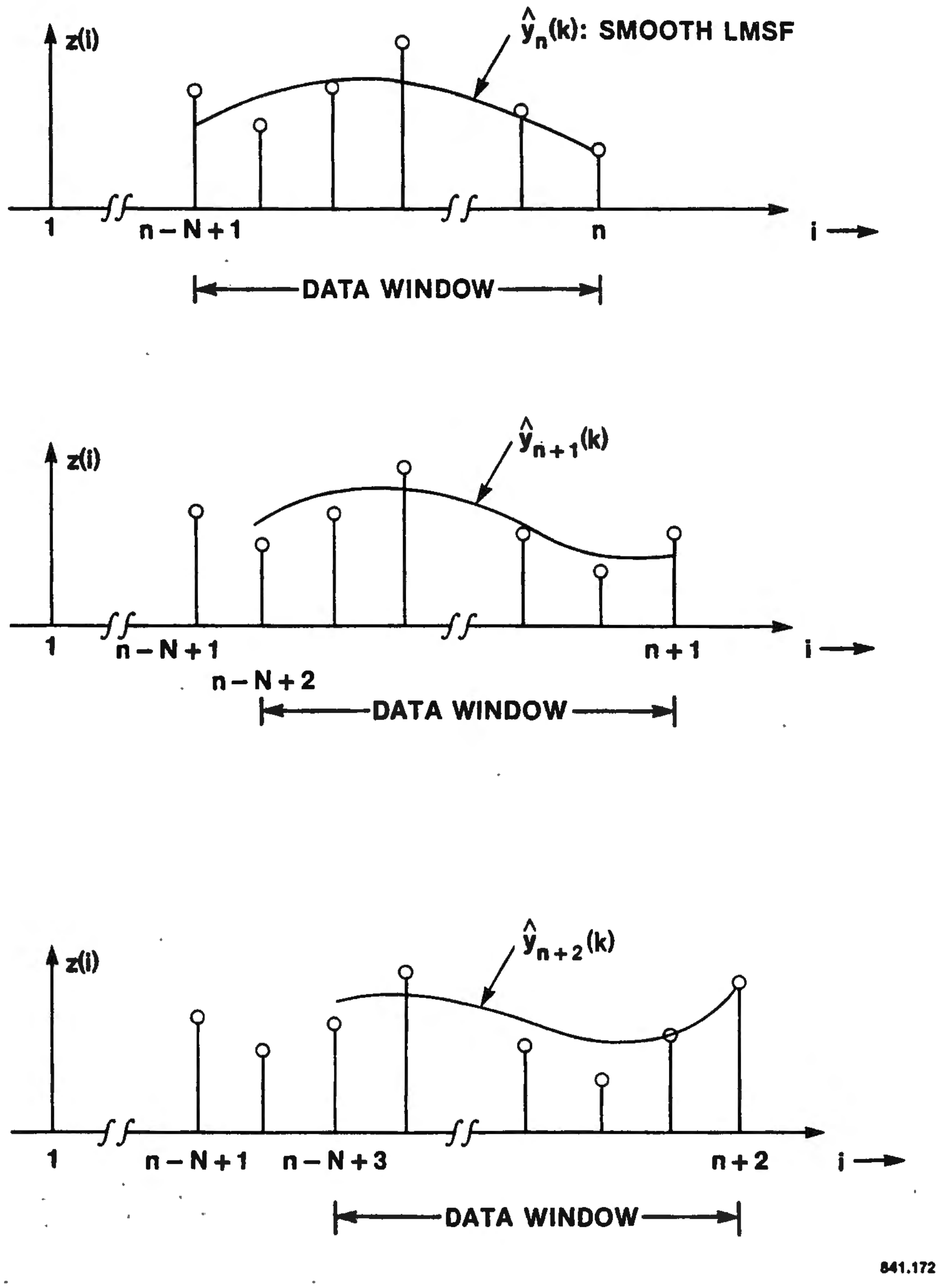

Figure 1. Moving Average Least Mean Square Fit

required for data lies within the operating window. Second is the apparent large computational requirement in comparison to the recursive implementation using Infinite Impulse Response filters [4]. The study presented here shows that while the size of storage requirement is unchanged, the computational load can be reduced significantly via an equivalent recursive formulation.

1.1 Problem Statement

Given a measurement sequence $(z(i))$; $i = 0, 1, 2, \ldots$, we denote the first N-point sequence by the column vector

$$\underline{z}_N = (z(1), z(2), \ldots, z(N))^T \tag{1}$$

where $(\ )^T$ denotes the vector transpose. It is desired to find a vector of Mth order polynomial coefficients

$$\underline{a}_N = (a_N(0), a_N(1), \ldots, a_N(M))^T \tag{2}$$

that minimizes the quadratic

$$J(\underline{a}_N) = (\underline{z}_n - H\,\underline{a}_N)^T(\underline{z}_N - H\,\underline{a}_N) \tag{3}$$

where H is a N x (M + 1) dimension matrix given by

$$H = \begin{bmatrix} 1 & t_1 & t_1^2 & \cdots & t_1^M \\ 1 & t_2 & t_2^2 & \cdots & t_2^M \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ 1 & t_N & t_N^2 & \cdots & t_N^M \end{bmatrix} . \tag{4}$$

For convenience, we shall assume a uniform sampling interval of Δt seconds such that one can write

$$t_i = (i - 1)\Delta t \ ; \qquad i = 1, 2, 3, \ldots \quad . \tag{5}$$

Differentiating Equation (3) with respect to $\underline{a}_n$ and setting the result to zero yields the desired LMSF coefficient vector $\hat{\underline{a}}_N$ given by

$$\hat{\underline{a}}_N = (H^T H)^{-1} H^T \underline{z}_N \tag{6a}$$

$$= (H^T H)^{-1} \underline{x}_N \tag{6b}$$

where

$$\underline{x}_N \triangleq H^T \underline{z}_N = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ t_1 & t_2 & t_3 & \dots & t_N \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ t_1^M & t_2^M & t_3^M & \dots & t_N^M \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \cdot \\ \cdot \\ \cdot \\ z_N \end{bmatrix} \tag{7a}$$

$$= \left[\sum_{i=1}^{N} z_i \quad \sum_{i=1}^{N} t_i z_i \quad \dots \quad \sum_{i=1}^{N} t_i^M z_i \right]^T . \tag{7b}$$

Note that the LMSF polynomial

$$z_N(t) = \sum_{m=0}^{M} a_N(m) t^m \quad ; \qquad 0 \leq t \leq N\Delta t \tag{8}$$

can be used to interpolate values at any point inside the data window. And in particular at the sampling instants, we have

$$\hat{\underline{z}}_N = H \hat{\underline{a}}_N = H(H^T H)^{-1} H^T \underline{z}_N \tag{9a}$$

and from Equations (1) and (9a) we obtain for the kth smooth sample (or element in $\hat{\underline{z}}_N$)

$$\hat{z}(k) = (\text{kth row of } H)\hat{\underline{a}}_N \; ; \qquad 1 \leq k \leq N \tag{9b}$$

Also note the subtle differences between Equation (6a) and Equation (6b). First, $\underline{z}_N$ is an N dimensional vector whereas $\underline{x}_N$ is a M + 1 dimensional vector. For all practical purposes, N, the number of data points, is much larger than M, the order of the LMSF. Second, the matrix $(H^T H)^{-1} H^T$ in Equation (6a) is (M + 1) x N dimension, and the matrix $(H^T H)^{-1}$ in Equation (6b) is (M + 1) x (M + 1) dimension. Thus assuming that $\underline{z}_N$ and $\underline{x}_N$ are known, then computing $\underline{a}_N$ via Equation (6b) will result in saving by a factor of (M + 1)/N of the number of multiplications and adds. For example, for a typical value of N = 100 and M = 4, we have (M + 1)/N = .05. This is a significant reduction of computation. Of course, the factor (M + 1)/N is the least lower bound since additional computations are required to obtain $\underline{x}_N$ from $\underline{z}_N$. If $\underline{x}_N$ was obtained from $\underline{z}_N$ directly using Equation (7a), then no improvement is gained. Thus it is desired to obtain an efficient computation of $\underline{x}_N$ from $\underline{z}_N$. Consequently, we will obtain an efficient algorithm to calculate $\underline{a}_n$ for each moving average window of length N.

Working toward this goal, we generalize Equation (1) in writing

$$\underline{y}_n = \begin{cases} (z_1, z_2, \ldots, z_n)^T ; & n \le N \\ (z_{n-N+1}, \ldots, z_{n-1}, z_n)^T ; & n > N \end{cases} \tag{10}$$

as an N-point data window ending at the sampling instant t_n. Then the LMSF coefficient vector $\hat{\underline{a}}_n$ can be written from Equations (6) and (10) as

$$\hat{\underline{a}}_n = (H^T H)^{-1} H^T \underline{y}_n \tag{11a}$$

$$= (H^T H)^{-1} \underline{x}_n ; \qquad n = 1, 2, \ldots \quad . \tag{11b}$$

Note that given $\hat{\underline{a}}_n$ the smooth sequence $\hat{\underline{y}}_n$ can be computed from the relation

$$\hat{\underline{y}}_n = H \hat{\underline{a}}_n \quad . \tag{11c}$$

Therefore, the basic statement of our problem is to obtain an efficient methodology to calculate $\hat{\underline{a}}_n$ for all n via obtaining a fast recursion in computing the vector $\underline{x}_n$. The matrix $(H^T H)^{-1}$ is specified for a given window size and can be pre-computed. Thus matrix inversion is not required for each update.

Note that letting $A = (H^T H)^{-1} H^T$ in Equation (11a) and writing

$$\underline{\hat{a}}_n = (\hat{a}_n(0) \quad \hat{a}_n(1) \quad \ldots \quad \hat{a}_n(M))^T \quad , \tag{12}$$

Equation (11a) can be rewritten as

$$\begin{bmatrix} \hat{a}_n(0) \\ \hat{a}_n(1) \\ \cdot \\ \cdot \\ \cdot \\ \hat{a}_n(M) \end{bmatrix} = \begin{bmatrix} A_0(1) & A_0(2) & \ldots & A_0(N) \\ A_1(1) & A_1(2) & \ldots & A_1(N) \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ A_M(1) & A_M(2) & \ldots & A_M(N) \end{bmatrix} \begin{bmatrix} y_n(1) \\ y_n(2) \\ \cdot \\ \cdot \\ \cdot \\ y_n(N) \end{bmatrix} \quad . \tag{13}$$

Therefore, the (m + 1)th element of $\underline{\hat{a}}_n$ is

$$\hat{a}_n(m) = \sum_{i=1}^{N} A_m(i)\, y_n(i); \quad m = 0, 1, \ldots, M \quad . \tag{14}$$

From Equation (14) we note that each element of $\underline{\hat{a}}_n$, say $\hat{a}_n(m)$, is the output of a Finite Impulse Response (FIR) filter operating on the data sequence. Written in the z-transform notation, the FIR filters are given by

$$A_m(z) = \sum_{i=0}^{N-1} A_m(i)\, z^{i-N}; \qquad m = 0, 1, \ldots, M \quad . \tag{15}$$

Equation (15) demonstrated the assertion made earlier about the LMSF polynomial coefficients.

A final point we want to make is that the LMSF coefficient $\hat{a}_n(m)$ corresponds to the mth order derivative of the LMSF polynomial evaluated at time $t = t_1 = 0$.

1.2 Technical Objectives

The primary objective of this memorandum is to present the step-by-step procedure that leads to the derivation of the fast-moving average least mean square recursion algorithm. The performance of the algorithm will be verified through extensive computer simulation. Machine timings for the recursive formulation will be obtained and used to compare with the corresponding batch approach.

1.3 Report Organization

The organization of this report is as follows. Section 2 presents the fast moving average linear least square recursion algorithm. Section 3 extends Section 2 results to the general least square case. Section 4 discusses the results in computer simulations and timings. Finally, Section 5 summarizes the results presented in this report.

2. MOVING AVERAGE RECURSIVE LINEAR LEAST SQUARE

In this section, we concentrate on the recursive algorithm development to the linear LMSF. Partly because this is one of the most important and widely used cases, and partly because the study on the linear case will serve as a natural base for extension to the general case.

2.1 Problem Formulation

Let $(z(i))$; $i = 1, 2, \ldots$ be the measurement sequence and $t_i = (i - 1)\Delta t$ be the corresponding sampling time where Δt is the sampling interval. A moving average first order fit of the form

$$z(t) = a_0 + a_1 t \tag{16}$$

with an N-point data window is desired. We seek an efficient recursion to calculate the LMSF coefficients $\underline{\hat{a}}_n = (\hat{a}_n(0), \hat{a}_n(1))^T$ for $n = 1, 2, 3, \ldots$

based on the N-point data sequence vector $\underline{y}_n$ which was defined earlier in Equation (10).

2.2 Result Derivation

Suppose the first N data points are available, then using Equations (4) and (11), we obtain

$$\hat{\underline{a}}_N = (H^T H)^{-1} H^T \underline{y}_N$$

$$= \begin{bmatrix} N & \sum_{i=1}^{N} t_i \\ \sum_{i=1}^{N} t_i & \sum_{i=1}^{N} t_i^2 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ t_1 & t_2 & t_3 & \dots & t_N \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ \cdot \\ \cdot \\ \cdot \\ z_N \end{bmatrix}$$

$$= \frac{1}{D} \begin{bmatrix} \sum_{i=1}^{N} t_i^2 & -\sum_{i=1}^{N} t_i \\ -\sum_{i=1}^{N} t_i & N \end{bmatrix} \begin{bmatrix} x_N(1) \\ x_N(2) \end{bmatrix} \tag{17}$$

where

$$D = N \sum_{i=1}^{N} t_i^2 - \left(\sum_{i=1}^{N} t_i \right)^2 \tag{18a}$$

$$x_N(1) = \sum_{i=1}^{N} z_i \tag{18b}$$

and

$$x_N(2) = \sum_{i=1}^{N} t_i \, z_i \tag{18c}$$

For uniform sampling such that $t_i = (i - 1)\Delta t$, the following quantities can be obtained.

$$\sum_{i=1}^{N} t_i = \Delta t \, \frac{N(N - 1)}{2} \tag{19a}$$

$$\sum_{i=1}^{N} t_i^2 = \Delta t^2 \, \frac{N(N - 1)(2N - 1)}{6} \tag{19b}$$

$$D = \Delta t^2 \, N^2 \left(\frac{N^2 - 1}{12}\right) \tag{19c}$$

and Equation (17) can be reduced to

$$\begin{bmatrix} \hat{a}_N(0) \\ \\ \hat{a}_N(1) \end{bmatrix} = \begin{bmatrix} \frac{2(2N - 1)}{N(N + 1)} & - \frac{6/\Delta t}{N(N + 1)} \\ \\ - \frac{6/\Delta t}{N(N + 1)} & \frac{12/\Delta t^2}{N(N^2 - 1)} \end{bmatrix} \begin{bmatrix} x_N(1) \\ \\ x_N(2) \end{bmatrix} \tag{20}$$

or equivalently, one can write

$$\underline{\hat{a}}_N = B \, \underline{x}_N \tag{21}$$

where the matrix $B \overset{\Delta}{=} (H^T H)^{-1}$ is a function of the data window length only. We shall next develop the recursion for $\underline{x}_n$ as follows.

Let $n = N + \ell$ where ℓ is an integer such that for any given integer N one can write $n = 1, 2, \ldots$. Then from Equation (18b) we obtain

$$x_{N+\ell}(1) = \sum_{i=1}^{N} z_{i+\ell} \quad . \tag{22a}$$

From which we can write

$$x_{N+\ell}(1) - x_{N+\ell-1}(1) = \sum_{i=1}^{N} (z_{i+\ell} - z_{i+\ell-1})$$

$$= z_{N+\ell} - z_{\ell} \quad . \tag{22b}$$

Substituting $n = N + \ell$ in Equation (22b) and rearranging terms yields the desired recursion

$$x_n(1) = x_{n-1}(1) + (z_n - z_{n-N}); \qquad n = 1, 2, \ldots \tag{22c}$$

where $x_0(1) \triangleq 0$ and $z_{n-N} = 0$ for $n \leq N$.

Now from Equation (18c) we obtain

$$x_{N+\ell}(2) = \sum_{i=1}^{N} t_i \, z_{i+\ell} \tag{23a}$$

and so for $t_1 = 0$, we obtain

$$x_{N+\ell}(2) - x_{N+\ell-1}(2) = \sum_{i=1}^{N} t_i (z_{i+\ell} - z_{i+\ell-1})$$

$$= \sum_{i=1}^{N-1} (t_i - t_{i+1}) z_{i+\ell} + t_N \, z_{N+\ell}$$

$$= -\Delta t \sum_{i=1}^{N} z_{i+\ell} + N\Delta t \; z_{N+\ell}$$

$$= -\Delta t \; x_{N+\ell}(1) + N\Delta t \; z_{N+\ell} \quad . \tag{23b}$$

Again substituting $n = N + \ell$ in Equation (23b) yields the desired recursion.

$$x_n(2) = x_{n-1}(2) + N\Delta t(z_n - x_n(1)/N) \tag{23c}$$

where $x_n(1)$ is obtained from Equation (22c) and $x_0(2) \overset{\Delta}{=} 0$.

2.3 Results Analysis

Thus Equations (22c) and (23c) define the recursions for the moving average linear least square. To calculate $\underline{x}_n$ from $\underline{x}_{n-1}$, the batch approach requires a total of 2N sums and N multiplications (Equations (22a) and (23a)), whereas the recursive formulation requires only four sums and one multiplication (Equations (22c) and (23c)). Note that the latter is independent of the window length.

In the following section, we will extend the recursive approach to the more general case.

3. MOVING AVERAGE RECURSIVE GENERAL LEAST SQUARE

In this section we generalize the results obtained for the linear least square problem to the general least square case. Analytic expression will be given for the reduction in computation.

3.1 Problem Formulation

Let $(z(i))$; $i = 1, 2, \ldots$ be the measurement sequence at time $t_i = (i - 1)\Delta t$. A moving average, N-point window, general LMSF using polynomial of the form

$$z(t) = a_0 + a_1 t + a_2 t^2 + \ldots + a_M t^M \tag{24}$$

is desired. We seek an efficient recursion to calculate the LMSF coefficients $\underline{\hat{a}}_n = (\hat{a}_n(0), \hat{a}_n(1), \ldots, \hat{a}_n(M))^T$ for n = 1, 2, 3, ... based on the N-point data sequence vector $\underline{y}_n$ defined in Equation (10).

3.2 Result Derivation

Suppose the first N data points are available, then using Equations (4) and (11), we obtain

$$\begin{aligned} \underline{\hat{a}}_N &= (H^T H)^{-1} H^T \underline{y}_N \\ &= B\, \underline{x}_N \end{aligned} \tag{25}$$

where $B \triangleq (H^T H)^{-1}$ is a (M + 1) x (M + 1) dimension matrix independent of the data and

$$\begin{aligned} \underline{x}_N &= H^T \underline{y}_N \\ &= \begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ t_1 & t_2 & t_3 & \ldots & t_N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ t_1^M & t_2^M & t_3^M & \ldots & t_N^M \end{bmatrix} \begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_N \end{bmatrix} \\ &= \begin{bmatrix} \sum_{i=1}^{N} z_i \\ \sum_{i=1}^{N} t_i \; z_i \\ \vdots \\ \sum_{i=1}^{N} t_i^M \; z_i \end{bmatrix} . \end{aligned} \tag{26}$$

Thus the mth component of the vector $\underline{x}_N$ is

$$x_N(m) = \sum_{i=1}^{N} t_i^{m-1} z_i \; ; \qquad m = 1, 2, \ldots, M + 1 \quad . \tag{27}$$

We want to develop a recursion for $\underline{x}_n$ for n = 1, 2, ...

For m = 1 and 2, the recursions are identical with the linear least square expressions given in Equations (22c) and (23c) and for convenience restated below as:

$$x_n(1) = x_{n-1}(1) + (z_n - z_{n-N}) \tag{28}$$

$$x_n(2) = x_{n-1}(2) + N\Delta t(z_n - x_n(1)/N) \tag{29}$$

where $x_0(1) = x_0(2) = 0$ and $z_{n-N} = 0$ for $n \leq N$.

Using the same approach as in the linear case, the recursion for the general case (for m > 2) can be obtained as follows:

$$x_{N+\ell}(m) - x_{N+\ell-1}(m) = \sum_{i=1}^{N} t_i^{m-1} z_{i+\ell} - \sum_{i=1}^{N} t_i^{m-1} z_{i+\ell-1} \quad .$$

$$= \sum_{i=1}^{N} t_i^{m-1}(z_{i+\ell} - z_{i+\ell-1})$$

$$= \sum_{i=1}^{N-1} (t_i^{m-1} - t_{i+1}^{m-1})z_{i+\ell} + t_N \, z_{N+\ell} - t_1 \, z_\ell \tag{30}$$

Since $t_1 = 0$, $t_N = (N - 1)\Delta t$ and using the binomial expansion

$$(1 + x)^n = 1 + nx + \frac{n(n - 1)}{2_i} x^2 + \dots + x^n$$

$$= \sum_{j=0}^{n} C_j^{(n)} x^j \tag{31a}$$

where

$$C_j^{(n)} = \frac{n!}{(n - j)!\, j!} \tag{31b}$$

is the binomial coefficient, we can write

$$t_i^{m-1} - t_{i+1}^{m-1} = \Delta t^{m-1} \left((i - 1)^{m-1} - i^{m-1}\right)$$

$$= \Delta t^{m-1} \left[(i - 1)^{m-1} - (1 + (i - 1))^{m-1}\right]$$

$$= \Delta t^{m-1} \left((i - 1)^{m-1} - \sum_{j=0}^{m-1} C_j^{(m-1)} (i - 1)^j\right)$$

$$= -\Delta t^{m-1} \sum_{j=0}^{m-2} C_j^{(m-1)} (i - 1)^j \quad . \tag{31c}$$

Substituting Equation (31c) in Equation (30) and simplifying as follows yields

$$x_{N+\ell}(m) - x_{N+\ell-1}(m) = -\Delta t^{m-1} \sum_{i=1}^{N-1} \sum_{j=0}^{m-2} C_j^{(m-1)} (i - 1)^j z_{i+\ell} + t_N^{m-1} z_{N+\ell}$$

$$= -\Delta t^{m-1} \sum_{j=0}^{m-2} C_j^{(m-1)} \left(\Delta t^{-j} \sum_{i=1}^{N-1} t_i^j z_{i+\ell}\right) + t_N^{m-1} z_{N+\ell}$$

$$= -\Delta t^{m-1} \sum_{j=0}^{m-2} C_j^{(m-1)} \Delta t^{-j} \left(\sum_{i=1}^{N} t_i^j z_{i+\ell} - t_N^j z_{N+\ell}\right) + t_N^{m-1} z_{N+\ell}$$

$$= -\Delta t^{m-1} \sum_{j=0}^{m-2} C_j^{(m-1)} \Delta t^{-j} \left(x_{N+\ell}(j+1) - t_N^j z_{N+\ell}\right) + t_N^{m-1} z_{N+\ell}$$

$$= \left(t_N^{m-1} + \sum_{j=0}^{m-2} \Delta t^{m-j-1} C_j^{(m-1)} t_N^j \right) z_{N+\ell} - \sum_{j=0}^{m-2} \Delta t^{m-j-1} C_j^{(m-1)} x_{N+\ell}(j+1)$$

$$= \left(\sum_{j=0}^{m-1} \Delta t^{m-j-1} C_j^{(m-1)} t_N^j \right) z_{N+\ell} - \sum_{j=0}^{m-2} \Delta t^{m-j-1} C_j^{(m-1)} x_{N+\ell}(j+1)$$

$$= \alpha_m z_{N+\ell} - \sum_{j=0}^{m-2} \beta_j^{(m-1)} x_{N+\ell}(j+1) \qquad (32a)$$

where α_m and $\beta_j^{(m-1)}$ are defined by the relations

$$\alpha_m = \sum_{j=0}^{m-1} \Delta t^{m-1} (N - 1)^j \; C_j^{(m-1)} \qquad (32b)$$

$$\beta_j^{(m-1)} = \Delta t^{m-j-1} \; C_j^{(m-1)} \quad . \qquad (32c)$$

Finally letting $n = N + \ell$ in Equation (32a) yields

$$x_n(m) = x_{n-1}(m) + \alpha_m \left(z_n - \frac{1}{\alpha_m} \sum_{j=0}^{m-2} \beta_j^{(m-1)} \; x_n(j+1) \right) \qquad (33)$$

as the general recursion for $m > 2$ with $x_0(m) = 0$ for all m.

3.3 Result Analysis

Equations (28), (29) and (33) describe the recursion for computing $\underline{x}_n$ given $\underline{x}_{n-1}$. Note that in Equation (33) the coefficient α_m and $\beta_j^{(m-1)}$ can be pre-computed and tabulated. Note also that in order to compute $x_n(m)$, one must first compute $x_n(1)$, $x_n(2)$, ..., $x_n(m-1)$ sequentially.

The following development briefly examines the total multiplications and adds required for the recursive approach. A study on the computation requirement in Equations (28), (29) and (33) shows that for each $x_n(m)$, the total multiplications and adds is bounded below by m. Therefore, for an Mth order LMSF, each $\underline{x}_n$ update requires at least (M+1)(M+2)/2 multiplications and adds. To update the polynomial coefficient vector $\hat{\underline{a}}_n$ there are additional $(M+1)^2$ multiplications and adds. Thus the total minimum number of multiplications and adds required for each $\hat{\underline{a}}_n$ update is

$$\begin{aligned} N_r &= (M + 1)^2 + (M + 1)(M + 2)/2 \\ &= (M + 1)(M + 1 + (M + 2)/2) \\ &= (M + 1)((3M + 4)/2) \quad . \end{aligned} \tag{34}$$

However, for the batch LMSF, it was shown in Section 1.1 that the corresponding number is

$$N_b = (M + 1)N \quad . \tag{35}$$

Therefore the recursive approach reduces the computation load with respect to the batch approach by a factor

$$\gamma = \frac{N_r}{N_b} = \frac{3M + 4}{2N} \quad . \tag{36}$$

This number is slightly greater than the ideal lower bound (M + 1)/N shown in Section 1.1

4. NUMERICAL EVALUATION

In this section, we discuss the method of implementation, numerical accuracy, computer simulation procedure, and the result of computing timings between the batch and the recursive approaches.

4.1 Computational Method

Appendix A lists the computer program used to calculate the computational speed between the batch and the recursive approaches. The following discussions briefly highlight some of the implementation consideration and the method of operation.

To prepare for the recursion, several arrays must be initialized to contain the constants needed in the calculations. First the matrix $(H^T H)^{-1}$ is needed where the H matrix is the matrix previously defined. For convenience we assume a unity sampling interval and write the measurement matrix $H(i,j) = (i - 1)^{j-1}$ where the row variable i has constraints $1 \leq i \leq N$ where N is the length of the data window. The column variable j has constraints: $1 \leq j \leq M+1$ where M is the fit order desired. The multiplication of H^T and H is trivial but taking the inverse creates quite a numerical problem since $H^T H$ is progressively ill conditioned. A single precision inversion method will only provide satisfactory results for a linear or first order fit; double precision will yield acceptable results for second and third order fits, but after third order very complicated methods will be necessary. It should be pointed out, however, that the matrix inversion needs to be calculated only once. Thus accurate numerical techniques can be used.

The $C_j^{(m)}$ coefficients are now calculated using the formula

$$C_j^{(m)} = \frac{m!}{(m - j)!\ j!}$$

where M is the fit order and j has constraints $0 < j < m - 2$. Only the last $M - 1$ rows of the $(M + 1) \times (M + 1)$ matrix are calculated since the first two rows are never used (see Appendix A).

The α_m vector is calculated next using the formula (see Equation (32b))

$$\alpha_m = \sum_{j=0}^{m-1} (N - 1)^j \ C_j^{(m-1)}$$

where N is the length of the data window used.

Once these constants have been prepared, the processing loop is entered where the recursive polynomial coefficient calculations are made.

Data is first read in and then the y_N vector (this is the vector $\underline{x}_N$ in Equation (33)) is calculated using the $C_j^{(m)}$ matrix and the α_m vector. y_N is then multiplied by the $(H^T H)^{-1}$ matrix to obtain the final polynomial coefficients. Next an estimate of the next path point is calculated using the first position outside of the data window as the target point. Once this is done, the oldest data point is discarded and is replaced by the newest piece of data. A pointer moves through a fixed vector to keep track of the data at the back of the data window (oldest data point) so that it can be used in the next recursion. The last step is to make the current recursion results "old" by setting the $y_{O(ld)}$ vector equal to the $y_{N(ew)}$ vector. This loop continues indefinitely or until a desired condition is met.

4.2 Numerical Simulation

The testing of this algorithm was done on DEC's VAX 11/780 located at the NUSC Code 321 Advanced Sonar Development Facility (ASDF). Fortran IV was used in its programming. For uniform testing, the same set of 10,000 data points were used for the recursive and batch-moving average LMSF. This input data consisted of a unit sampling, linearly increasing ramp with unity slope as the signal and an additive, zero mean, unit variance noise sequence.

Using a window length of 50 seconds (i.e., N = 50), the fitting performance of the moving average recursive LMSF and the absolute error characteristics as a function of time are presented in Figures 2 to 5 for the first, second, third, and fourth order LMSF, respectively. There are two important aspects which need to be pointed out. First, although not shown, extensive testings show that when the data window is full, both batch and recursive approaches yield identical results. We should caution the reader that this may not be the case when a numerical problem exists in matrix inversion. For example, when the matrix $H^T H$ is singular, the recursion approach fails while there exists other techniques such as the singular value decomposition (SVD) in solving the batch approach. Second, prior to filling the data window

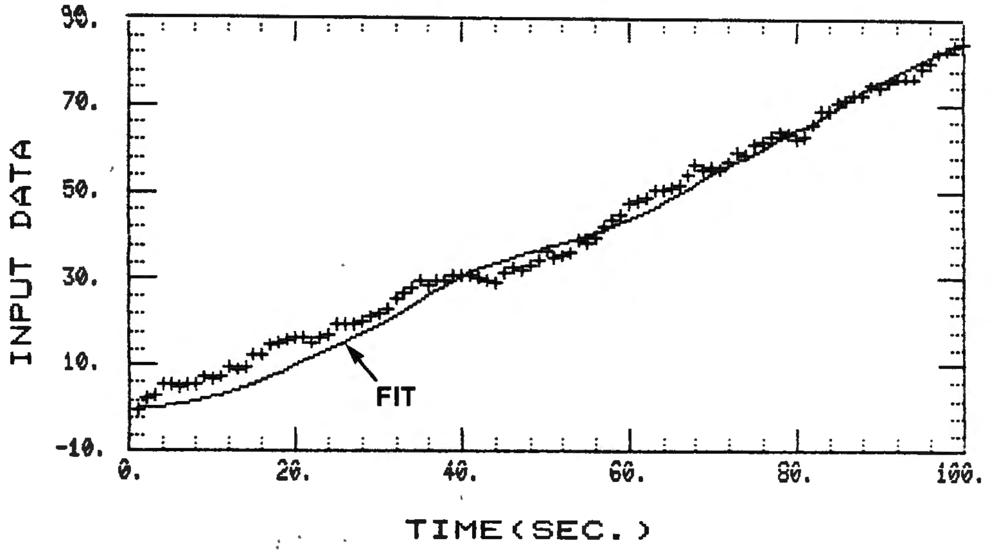

Figure 2a. Recursive Moving Average First Order LMSF (Window Length = 50)

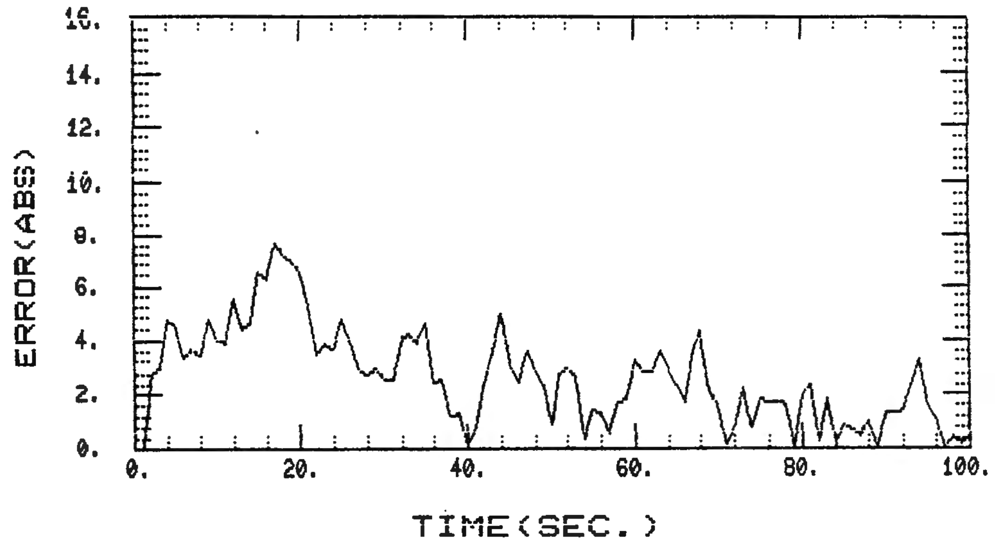

Figure 2b. Recursive Moving Average First Order LMSF Absolute Error

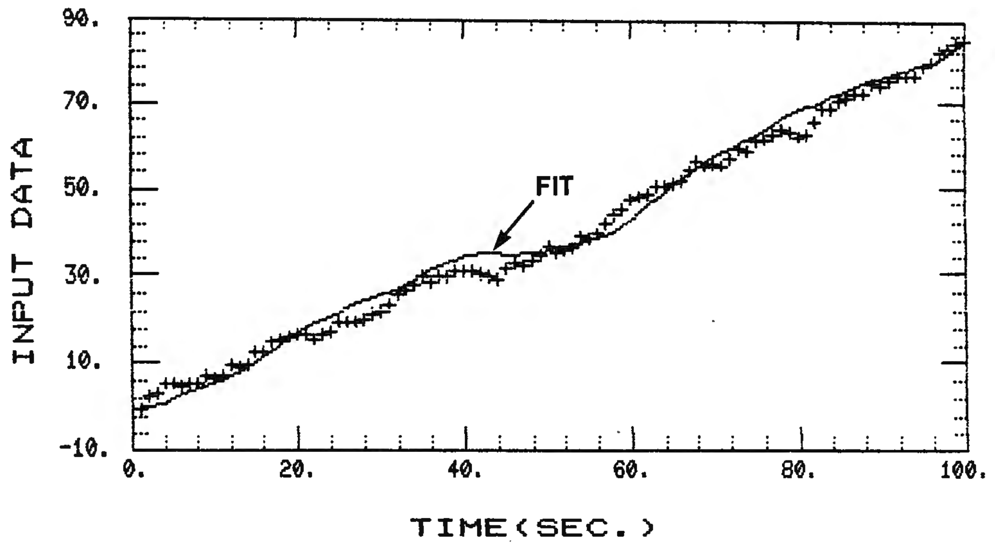

Figure 3a. Recursive Moving Average Second Order LMSF (Window Length = 50)

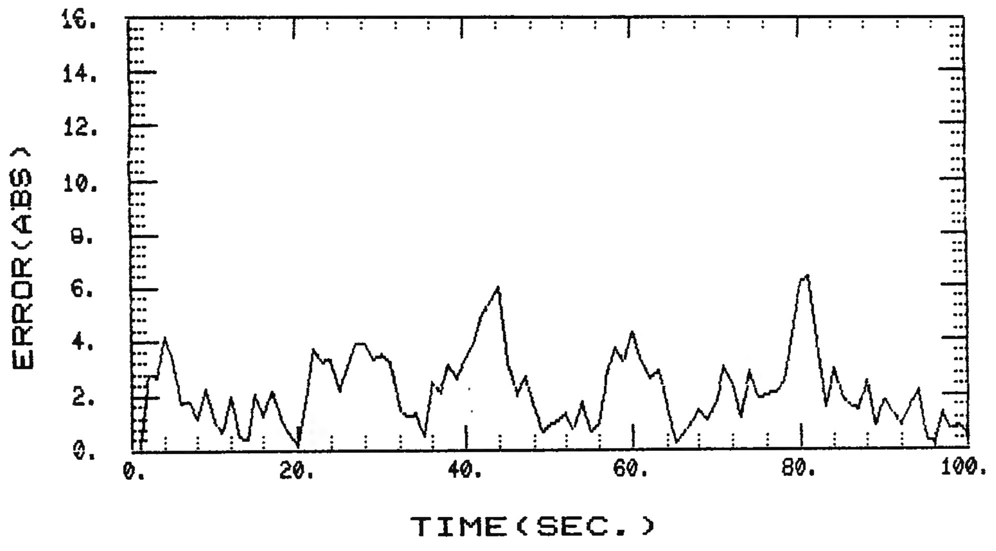

Figure 3b. Recursive Moving Average Second Order LMSF Absolute Error

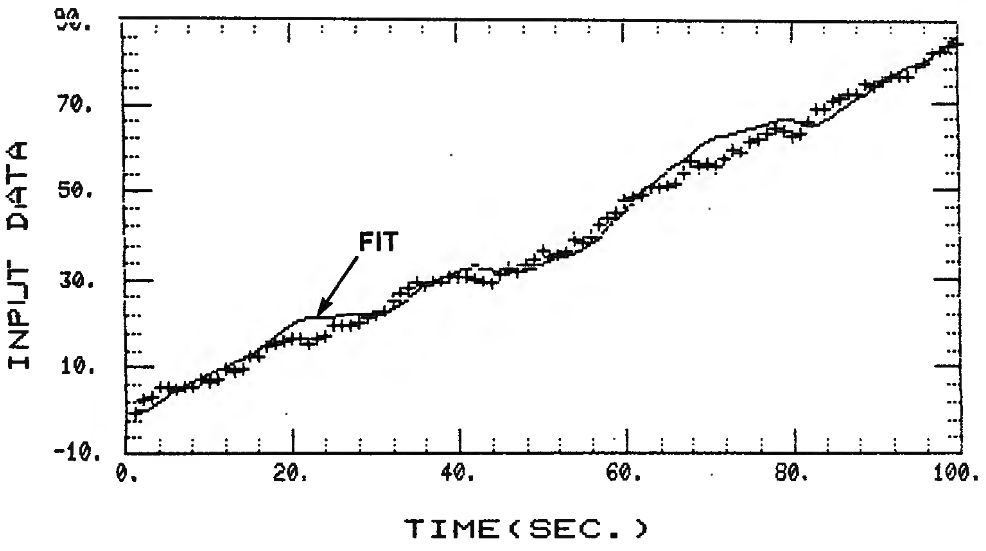

Figure 4a. Recursive Moving Average Third Order LMSF (Window Length = 50)

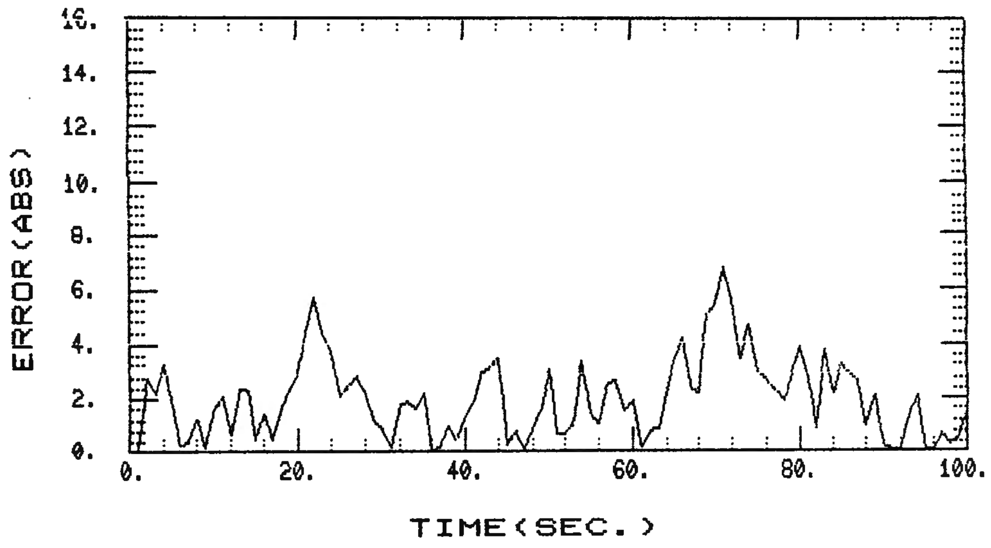

Figure 4b. Recursive Moving Average Third Order LMSF Absolute Error

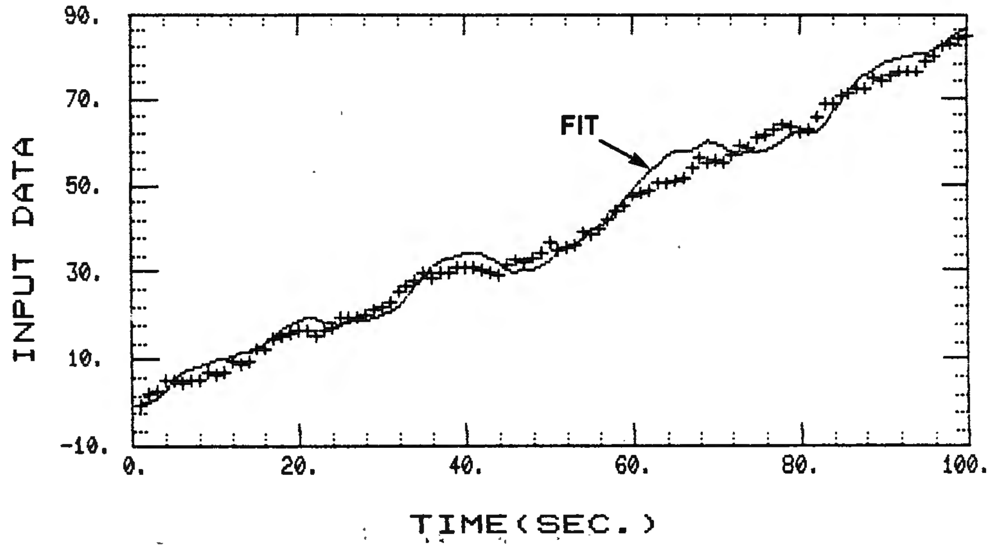

Figure 5a. Recursive Moving Average Fourth Order LMSF (Window Length = 50)

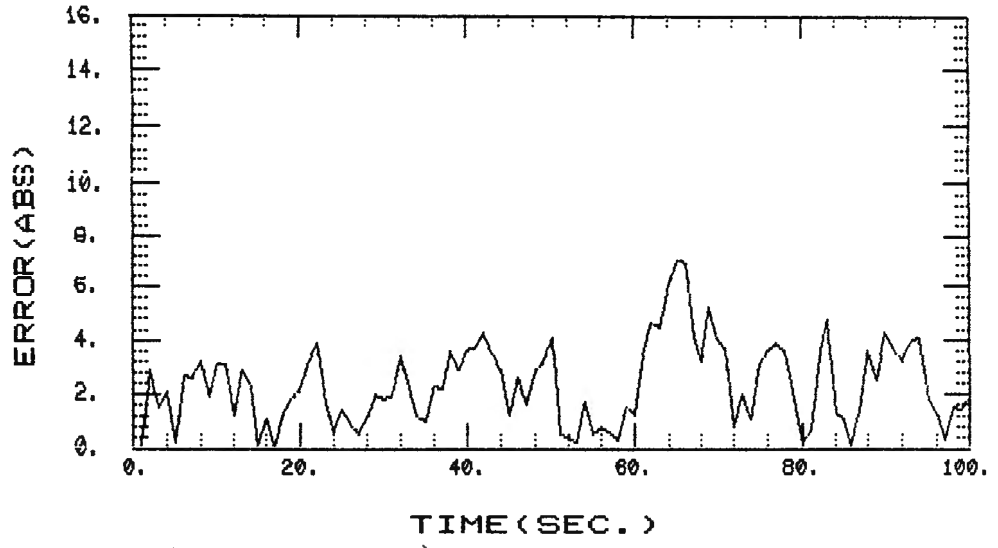

Figure 5b. Recursive Moving Average Fourth Order LMSF Absolute Error

(50 seconds), the recursive approach has been properly initialized and maintains relatively good fits during this interval. It is observed that higher order LMSF yield smaller fitting errors prior to the data window being fully filled. On the other hand, the first order LMSF gives the best result after the data window is filled. This is to be expected since the first order LMSF matches the input signal component. Although not shown, the batch approach, on the other hand, produces no output at all before the data window is full. Thus, this is the additional advantage for the recursive approach.

We next examine the relative computation time required for the batch and the recursive approaches. The CPU time has been selected as the performance measure variable. Using a number of different length windows, both approaches were used to run through the 10,000 data points with the resulting CPU times properly recorded. The results are presented in Figures 6 to 9. Close study of these results shows that (1) while the batch CPU time increases linearly with the window length, the recursive CPU time remains constant and insensitive to the window length as expected, (2) for a given window length, the batch CPU time increases (actually linearly) with the order of the LMSF, and (3) as window length shortens, CPU times between batch and recursive becomes comparable.

4.3 Speed Comparisons

We next obtain the speed ratio (SR) of recursive CPU time over batch CPU time. The results are shown in Figures 10 to 13 for first, second, third, and fourth order respectively. Also included are the lower bounds obtained based on the pure number of multiplications and adds required. For each update it was shown earlier that the batch approach requires approximately (M + 1)N number of operations. For the recursive, it was obtained exactly that there are

$$N_a = 1 + \frac{(M + 1)(3M + 2)}{2} \qquad (37)$$

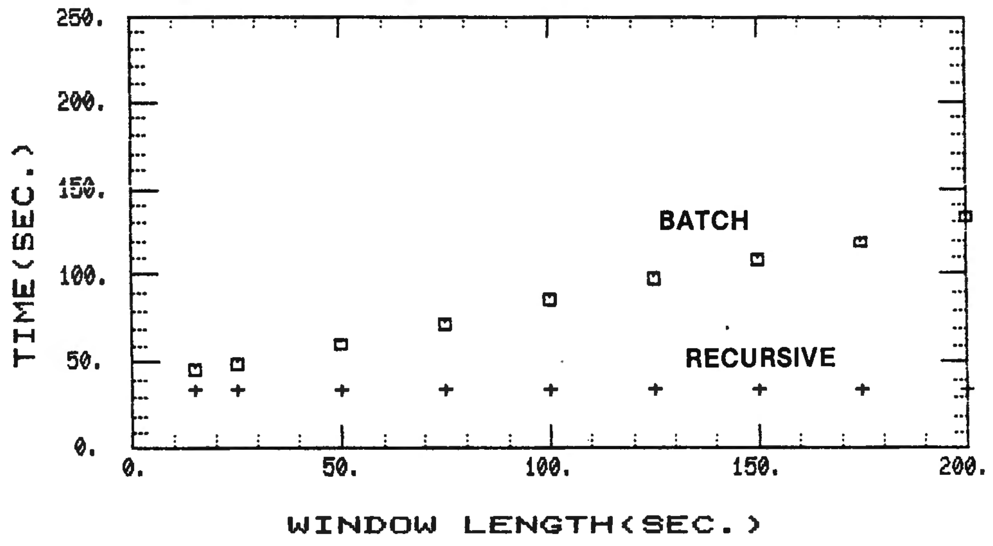

Figure 6. CPU Time Versus Window Length Between Batch and Recursive First Order LMSF

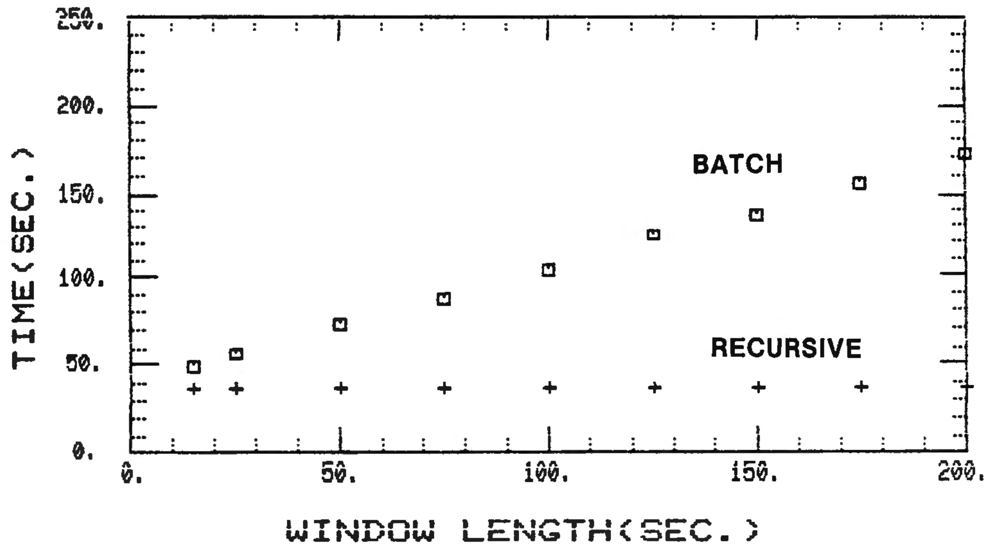

Figure 7. CPU Time Versus Window Length Between Batch and Recursive Second Order LMSF

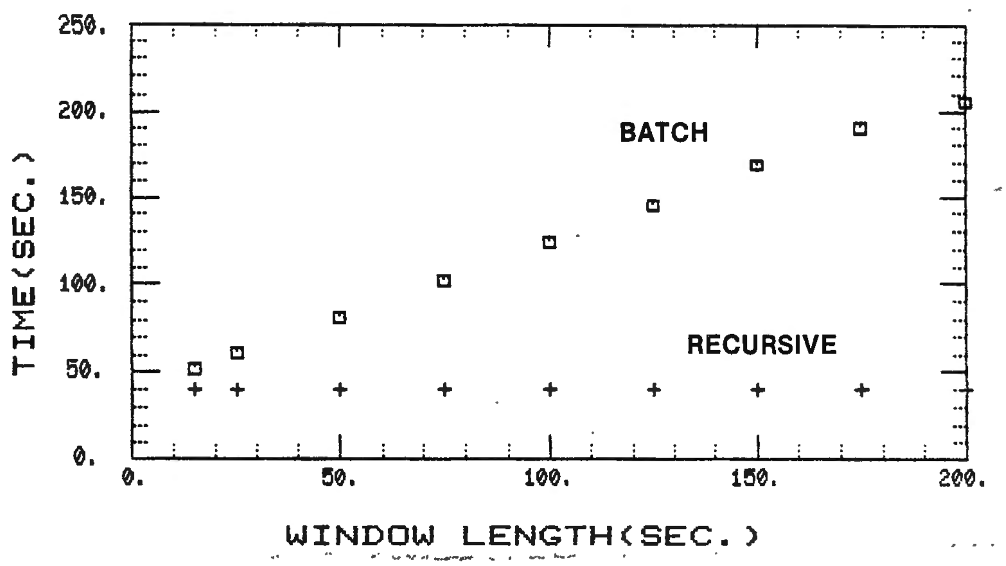

Figure 8. CPU Time Versus Window Length Between Batch and Recursive Third Order LMSF

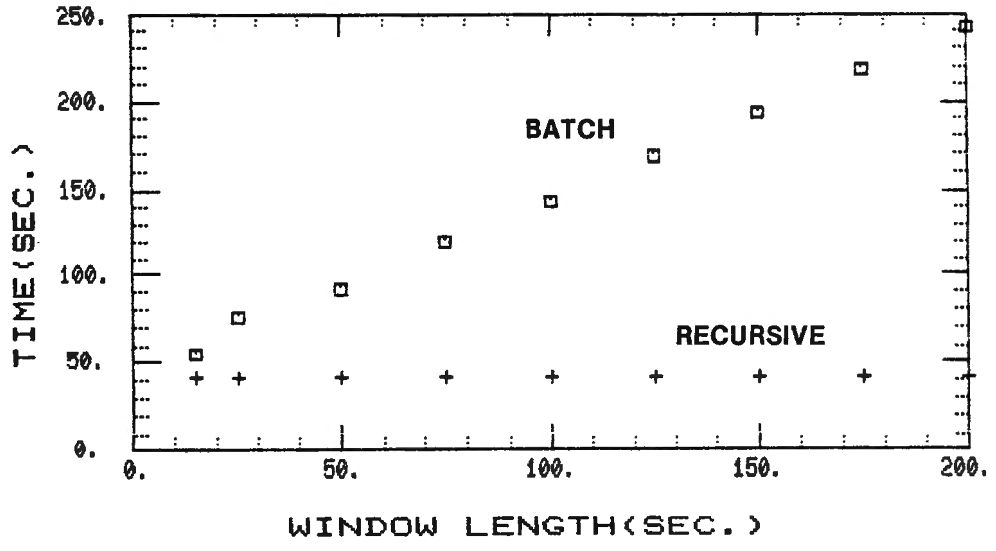

Figure 9. CPU Time Versus Window Length Between Batch and Recursive Fourth Order LMSF

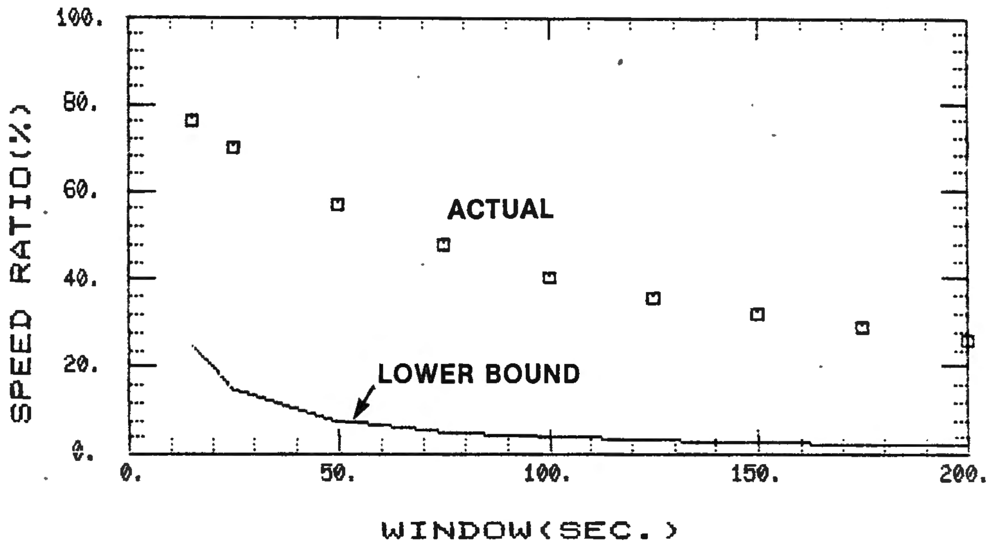

Figure 10. Recursive Over Batch Speed Ratio Versus Window Length (First Order LMSF)

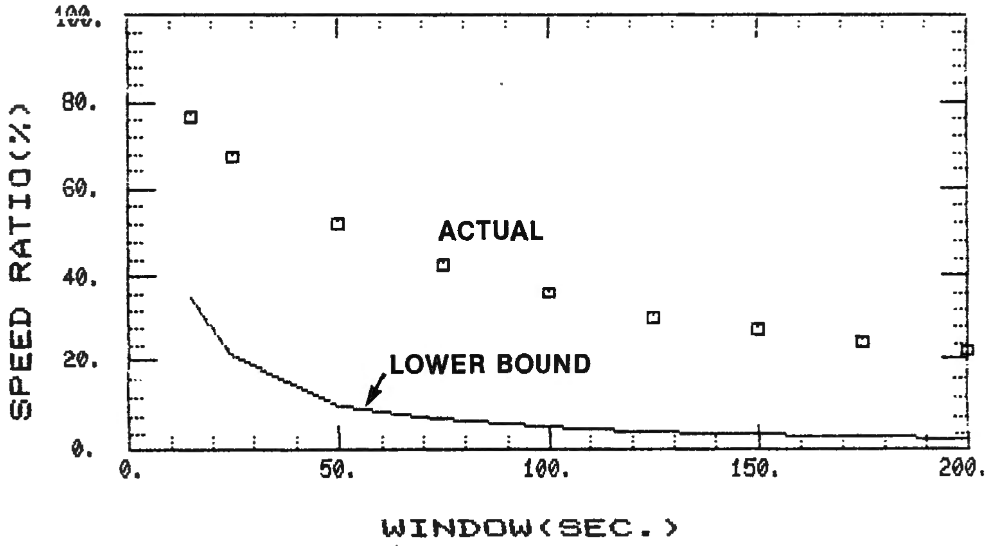

Figure 11. Recursive Over Batch Speed Ratio Versus Window Length (Second Order LMSF)

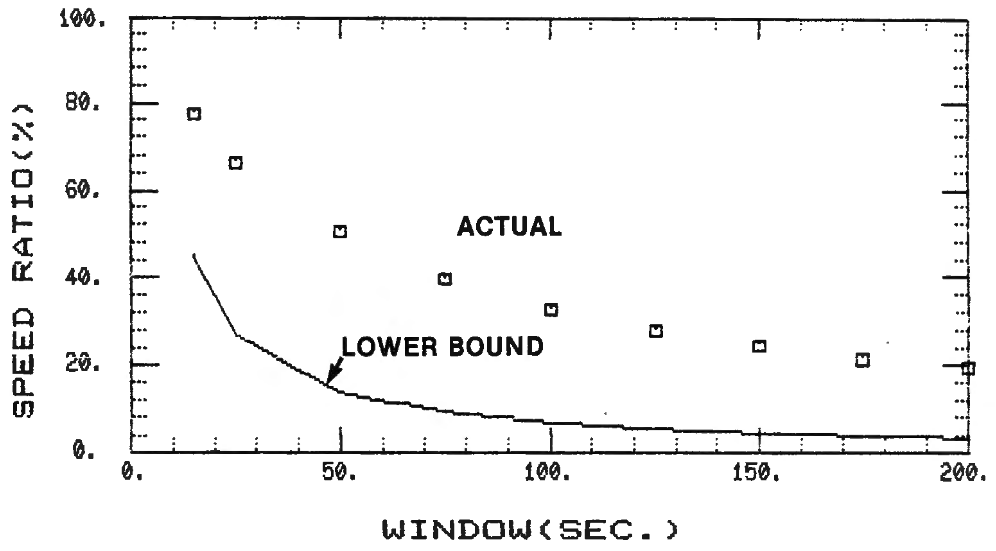

Figure 12. Recursive Over Batch Speed Ratio Versus Window Length (Third Order LMSF)

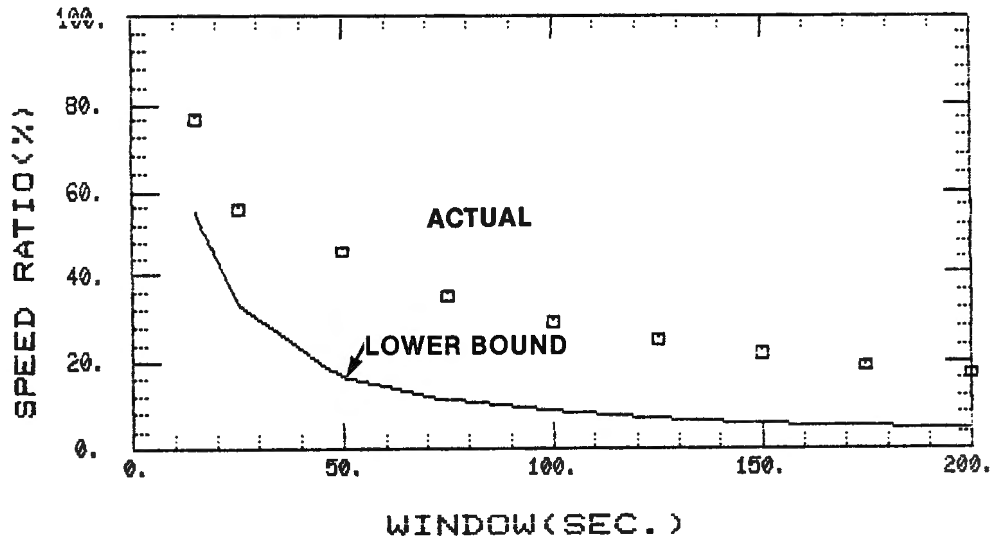

Figure 13. Recursive Over Batch Speed Ratio Versus Window Length (Fourth Order LMSF)

number of adds (Note, M is the order of the fit) and

$$N_m = \frac{(M + 1)(3M + 4)}{2} - 1 \tag{38}$$

number of multiplications. Let T_a and T_m be the time required for a single add and multiplication, respectively. Then the lower bound (LB) is computed from the formula

$$LB = \frac{N_a T_a + N_m T_m}{(M + 1)N T_a T_m} \tag{39}$$

Examining Figures 10 to 13 shows that the actual SRs fall off as a function of window length at a rate similar to the lower bound. The significant difference between the actual SR and the LB is due to programming overhead cost; i.e., CPU time for non-arithmetic operations. With more efficient programming, this difference is expected to reduce. At any rate, Figures 10 to 13 show that at a window length of 200, the recursive CPU time is only 20 percent of the batch CPU time. The corresponding number for the lower bound is 4 percent. Note also that in contrast to the lower bound behavior, Figures 10 to 13 show that for a given window length N the actual SRs fall off as a function of M, the order of the fit. This is the case because the increases in actual overhead is faster for the batch than the recursive as a function of the window length.

4.4 Discussion of Results

As previously shown, the recursive method provides results as accurately as the batch method with much less computational speed and effort. In addition, the recursive method provides a useful estimate even before the data window is filled.

The theoretical speed advantage shows the great improvement of this method over the batch method. The actual measured speed comparison may be made to move toward this ideal SR by optimizing the code used in programming.

5. SUMMARY AND CONCLUSIONS

This study developed an efficient recursive algorithm to implement a moving average Least Mean Square Fit (LMSF) procedure. The following briefly summarizes the significant findings of this study.

1. A recursive formulation of a moving average LMSF can be implemented with a theoretical ratio in computation speed gain (recursive over batch) of $(3M + 4)/2N$, where M is the order of the LMSF and N is the window length.

2. It was shown in Section 3 that recursive formulation is identical to the batch in terms of filter memory length and data storage requirement.

3. Ignoring the potential singular value inversion and other numerical problems, recursive moving average LMSF gives outputs identical to the batch LMSF when the data window is filled. However, prior to attaining the full data window, the recursive approach has the additional advantage that it could also provide meaningful outputs if good a priori information is used to initiate the recursion.

4. The recursive and the batch LMSF have been implemented and tested extensively on a VAX 11/780 computer. The measured performance in terms of speed and output behavior agree well with the analysis. (Detailed discussions of the simulation results were presented in Section 4.)

5. The significant reduction in computational load for the moving average LMSF could make it applicable for real-time processing.

6. The moving average LMSF can be used for a wide variety of applications in data smoothing, noise filtering, and numerical interpolation.

7. Although this study developed the moving average recursive LMSF assuming a time data sequence, the result, however, is applicable to any other sequences such as frequencies and wave numbers.

8. We have shown the speed advantage of the recursive approach. However, we have ignored the relative comparison from the numerical point of view. This important comparison should be addressed in a future study.

REFERENCES

1. L. C. Ng and R. A. LaTourette, "Equivalent Bandwidth of a General Class of Polynomial Smoothers", J. Acoust. Soc. Am. 74(3), September 1983. Also NUSC Technical Report TR 6601, dated 19 July 1982.

2. A. Gelb, "Applied Optimal Estimation", the M.I.T. Press, Cambridge, MA, 1974.

3. N. Morrison, "Introduction to Sequential Smoothing and Prediction", McGraw-Hill Book Company, 1969.

4. B. A. Bowen and W. R. Brown, "Signal Processing and Signal Processors", Prentice-Hall, Inc., 1982.

APPENDIX A

LISTINGS OF COMPUTER PROGRAM

```
C   RECURSIVE METHOD FOR FITTING A POLYNOMIAL TO DATA
C    -UNIFORM SAMPLING ASSUMED-

      REAL*8 C(10,10),CM(8),YN(10),YO(10),AN(10,1),Z(250),
     &H(250,10),HTET(10,10),HTH(10,10),HT(10,250),HI(10,10)
      IDGT=0
      WRITE (6,3)
 3    FORMAT (' INPUT SYSTEM ORDER (0<= M <=9)')
      READ *,M
      WRITE (6,6)
 6    FORMAT (' INPUT THE LENGTH OF THE DATA WINDOW (1< LW <250)')
      READ *,LW

      OPEN(UNIT=10,NAME='OUT.DAT',TYPE='OLD',FORM='FORMATTED')

      DO I=1,LW
         Z(I)=0.0
      END DO

      DO I=1,M+1
         YO(I)=0.0
         AN(I,1)=0.0
      END DO

C   CREATE THE H MATRIX

      DO I=1,M+1
         DO K=1,LW
            IF (I-1.EQ.0) THEN
               H(K,I)=1
            ELSE
               H(K,I)=(K-1)**(I-1)
            END IF
         END DO
      END DO

C   CREATE H TRANSPOSE

      DO I=1,LW
         DO J=1,M+1
            HT(J,I)=H(I,J)
         END DO
      END DO

C   CREATE (HTH)**(-1)

      CALL M1MULT (HT,H,HTH,M+1,LW,M+1)
      DO I=1,M+1
         DO J=1,M+1
            HI(I,J)=HTH(I,J)
         END DO
      END DO
      CALL INVERT(HI,M+1)
      DO I=1,M+1
         DO J=1,M+1
            HI(I,J)=HI(I,J)*LW
         END DO
      END DO

C   CALCULATE MATRIX OF BINOMIAL COEFFICIENTS

      IF (M.GT.1) THEN
         DO I=2,M
           C(I-1,1)=1
         END DO
         DO L=2,M
            C(L-1,L+1)=1
            K1=1
            DO I=2,L
               K1=K1*I                      !    K1=(M-1)!
            END DO
            DO J=2,L
               K2=1
               K3=1
               DO N=1,(L-J+1)
                  K2=K2*N                   !    K2=(M-I)!
               END DO
               DO N=2,J
                  K3=K3*(N-1)               !    K3=(I-1)!
               END DO
               C(L-1,J)=K1/K2/K3
            END DO
         END DO
      END IF

C   CALCULATE CM VECTOR

      IF (M.GT.1) THEN
         DO I=3,M+1
            DO J=1,I
               CM(I-2)=CM(I-2)+(LW-1)**(J-1)*C(I-2,J)
            END DO
            CM(I-2)=CM(I-2)/FLOAT(LW)
         END DO
      END IF

C   BEGIN POLYNOMIAL MATCHING RECURSION

      IP=1
      L=0

      DO WHILE(L.LT.1)

C    READ IN NEW DATA

         READ (10,*) ZA
         WRITE (9,*) EST,ZA

         IF (ZA.EQ.0.) GOTO 100

         YN(1)=YO(1)+(ZA-Z(IP))/FLOAT(LW)
         IF (M.GT.0) THEN
            YN(2)=YO(2)+ZA-YN(1)
            IF (M.GT.1) THEN
```

```
                 DO I=3,M+1
                    R=Ø
                    DO J=1,I-1
                       R=R+C(I-2,J)*YN(J)
                    END DO
                    YN(I)=YO(I)-R+CM(I-2)*ZA
                 END DO
              END IF
           END IF

C     CALCULATE POLYNOMIAL COEFFICIENTS

           CALL M3MULT (HI,YN,AN,M+1,M+1,1)

C     CALCULATE ESTIMATE OF NEXT POINT

           EST=Ø
           DO J=1,M+1
              EST=EST+AN(J,1)*(LW+1)**(J-1)
           END DO

C     UPDATE DATA WINDOW

           Z(IP)=ZA
           IP=IP+1
           IF (IP.GT.LW) IP=1

C     MAKE CURRENT RECURSION RESULTS 'OLD'

           DO I=1,M+1
              YO(I)=YN(I)
           END DO
        END DO

 1ØØ    STOP
        END
```

DISTRIBUTION LIST

Internal

02111	A. H. Lotring
021311	(NLON) (3)
021312	(Dist. Center) (10)
03	
0302	G. M. Hill
039	J. B. Hall J. M. Bradshaw
10	W. A. Von Winkle
101	A. J. VanWoerkom K. Lima
32	F. J. Kingsbury
3202	T. O. Fries D. B. Stockton E. B. Siborg C. Drenzewski
321	R. Garber D. J. Harrington
3211	J. K. O'Sullivan N. L. Owsley D. Pistachio R. Tozier L. Meier E. Walters R. R. Kneipfer J. W. Fay G. R. Swope
3212	S. A. Dzerorvych P. J. Barnikel H. Teller W. Zwolinski B. Buehler R. I. Hayford J. P. Iannielo F. H. Maltz
3213	S. M. Kessler W. H. Axtell G. F. Drury W. K. Fischer L. Lampert R. A. LaTourette L. C. Ng (10) J. V. Sanchis W. A. Struzinski P. Puterio E. Lowe J. Lee P. Lambert (5) R. Cox S. Greineder S. Okur
325	
3251	
3251	G. Volkov J. Barkley
3291	J. R. Pratt P. Dean J. E. Brown J. DePrimo
33A	B. Cole P. Herstein G. M. Mayer
33B3	C. Nawrocki
3312	M. M. Hundt
3314	G. C. Carter J. P. Beam R. F. Dwyer R. L. Mason A. H. Quazi A. H. Nuttall I. Cohen K. Scarbrough

DISTRIBUTION LIST

Internal (Cont'd)

35	
351	J. S. Davis B. W. Guimond
352	J. C. Hassab
35201	V. J. Aidala
3521	W. G. Ravo
3523	
3523	K. Gong J. Baylog
36	
363	J. R. Short
60	J. S. Cohen J. J. Dlubac

External

CNM, MAT-05B	Capt. Z. L. Newcomb
NAVSEA PMS-409	Capt. D. Bolka Dr. R. Snuggs Mr. M. Roberts Mr. W. Johnson Mr. R. McNamara Dr. Y. Yam
NAVSEA SEA-63R	Dr. D. Porter
COMSUBDEVRON 12	Lt. J. Halsema Cdr. T. Tehan
NRL	
NAVPGSCOL	
DARPA	
IBM Manassas, VA	Dr. G. Johnson Dr. D. Ohlm

External (Cont'd)

TRACOR TX	Mr. J. Wilkinson Dr. T. Leih
EG&G Manassas, VA	Dr. J. Hughen Mr. P. Bhrums
A&T N. Stonington, CT	Mr. H. Jarvis Mr. D. Ghen Mr. L. Gorham Dr. C. Wenk
UCONN Storrs, CT	Prof. C. Knapp Prof. C. Nikias Prof. Y. Bar-Shalom Prof. P. Luh Prof. D. Klienman
AIDS Mountain View, CA	Dr. C. Y. Chong
LLNL Livermore, CA	Dr. J. V. Candy Dr. C. J. Herget Mr. W. C. Ng